Vol.2012-ARC-199 No.8
2012/3/28

複数のGPUを用いた 3次元Smith-Waterman法の性能評価

須戸里織[†1] 吉見真聡[†1]
三木光範[†1] 廣安知之[†1]

近年，画像処理専用のハードウェアであるGPU（Graphics Processing Unit）を汎用計算に用いる動きが多くみられるようになってきている．GPUのような並列計算ハードウェアでの高速化が期待されているアプリケーションの中のひとつに，SW（Smith-Waterman）法がある．SW法はDNA配列の相同性検索に用いられているが，応用研究として時系列データの解析にも用いられており，そのひとつに，うつ病診断に用いられるfNIRSによる脳血流量の解析がある．fNIRSは多数の計測点から出力があり，2つの文字列の比較を行うSW法を利用して解析するためには，一度に比較できる系列数の拡張が課題となる．そこで本研究報告では，3次元SW法を提案し，fNIRSのデータに基づいて，処理能力をCPUとGPUで，またGPUの枚数を変化させ評価した．

A performance evaluation of three-dimensional Simith-Waterman method with multiple GPUs

Saori Sudo,[†1] Masato Yoshimi,[†1] Mitsunori Miki[†1]
and Tomoyuki Hiroyasu[†1]

Graphic processing units (GPUs), which was entirely used in image processing, have been applied to a lot of studies and products in general purpose computation. One of the most famous algorithm expected to accelerate by utilizing GPU is Smith-Waterman (SW) algorithm in bioinformatics. The algorithm is also applied to analysis of time-series data, such as cerebral blood flow detected by fNIRS which is used to make a diagnosis for depression. As fNIRS outputs data from a lot of measure points, handreds of executions of SW-algorithm are required to obtain relationships among channels. The expansion of the number of series which analyzed simultaneously may be a solution to increase the analysis quality. This report proposed the three-dimensional SW algorithm as a first step to address the problem, and evaluated performances on a CPU, a GPU, and multiple GPUs based on the actual data measured from fNIRS.

1. はじめに

近年，画像処理用のハードウェアであるGPU（Graphic Processing Unit）は，本来の利用目的だけでなく汎用計算に利用されるようになってきた．GPUを画像処理以外の汎用計算に応用することをGPGPU（General-purpose computing on GPU）[1]という．GPUはCPUと比べて，価格と消費電力量あたりの処理能力が高いことから注目を集めている．GPUは多くのコア数をもつため，条件分岐のない単純な計算を並列処理するのに性能を発揮するが，条件分岐を行う複雑な命令では処理能力が低下するという特徴がある．

SW（Smith-Waterman）法[2]は，生物学の分野においてDNA配列中で同じ順序で並んでいる配列を見つける配列アラインメント[3]の手法のひとつである．SW法は計算において細粒度並列性が高く，GPUやFPGAなどの並列計算ハードウェアで高速化が行われてきた．SW法はDNA配列の解析のほか，うつ病の発見に効果を発揮するfNIRS[4]のデータ解析への利用が考えられている[5]．SW法は部分的な類似部分を探索するのに適しているため，fNIRSの解析に向いているとされる．SW法は2つの文字列の比較を行うので，多数の時系列データが出力として得られるfNIRSの解析を行うには，全組み合わせについてSW法を実行しなければならない．効率的にSW法を用いてfNIRSのデータ解析を行うためには，SW法で同時に比較できる文字データの個数を増やす必要がある．また，SW法は比較する文字列が長くなるほど計算に時間がかかるため，GPUやFPGAなどで高速化を行う必要がある．

そこで本研究報告では，比較できるデータ数を3つに拡張した3次元SW法の提案を行い，GPGPU向けの統合開発環境であるCUDAを用いて，GPUで高速に計算する手法について述べる．また，提案したアルゴリズムをCPUとGPUで，また，GPUの枚数を変化させて比較検討を行う．

以降では，2章と3章でそれぞれSW法と3次元SW法について，4章でCUDAの概要について，5章で関連研究について述べる．次に，6章で本研究報告における実装を述べ

†1 同志社大学
Doshisha University

Vol.2012-ARC-199 No.8
2012/3/28

X \ Y	-	A	C	A	C
-	0	0	0	0	0
A	0				
G	0				
C	0				
A	0				

図 1 文字列テーブル

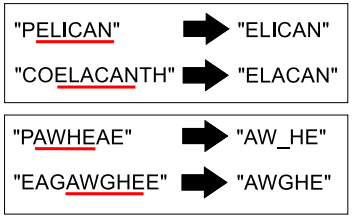

図 2 SW 法で抽出した類似部分列の例

て，7 章で処理性能について評価を行い，8 章でまとめを述べる．

2. Smith-Waterman 法

2.1 概　　要

SW 法は 2 つの文字列の共通部分列を抽出するための手法である．SW 法は部分的な類似部分を探索する，局所的ペアワイズ配列アラインメントのひとつである．SW 法は他の配列アラインメントと違い，類似度が最大の共通部分列のみでなく，複数の類似部分列を抽出することができる．

2.1.1 アルゴリズム

図 1 に SW 法の計算で使用する文字列テーブルを示す．SW 法では類似部分列を求めるために，図 1 に示すような対象文字列 X，Y を，行と列に割り当てた文字列テーブルを計算する．図 1 の左上の要素から右下の要素へ，注目している 2 つの文字がどの程度類似しているかを示すスコアを計算し値を格納していく．文字列テーブルの右下まで計算を終えたのち，最も高いスコアの要素からスコアが 0 になるまで，要素を選択した経路を辿り，類似文字列を抽出する．この操作をトレースバックという．SW 法によって抽出された類似文字列の例を**図 2** に示す．

2.2 計 算 手 順

SW 法のスコアの計算では match，mismatch，gap というパラメータを用いる．match は 2 つの文字列が一致したとき，mismatch は不一致のとき，gap はスペースの挿入に関するスコアである．SW 法では一般的に match = 1，mismatch = -1，gap = -1 が用いられる．DNA の解析においては，BLOSUM 行列と呼ばれる，2 つのアミノ酸間の相同性スコアを示すアミノ酸置換テーブルの値を用いるが[6]，本稿では時系列データの類似度を調べるため，アミノ酸置換テーブルは用いない．

長さが m の配列 $X = x_1x_2...x_m$ および長さが n である配列 $Y = y_1y_2...y_n$ を対象としたときの，SW 法のアルゴリズムの流れを以下に示す．

Step1. 配列 X，Y について，文字列テーブルを作成し，それぞれの文字列を列と行に割り当てる．

Step2. i 行 j 列にある要素が $SW(i,j)$ であるとすると，$0 \leq k \leq m$ および $0 \leq l \leq n$ において，$SW(k,l) = SW(k,0) = SW(0,l) = 0$ と初期化を行う．

Step3. それぞれの要素について式（1）を基にスコアを計算する．

$$SW(i,j) = \max \begin{cases} SW(i-1,j-1) + A \\ SW(i-1,j) + \text{gap} \\ SW(i,j-1) + \text{gap} \\ 0 \end{cases} \quad (1)$$

$$A : \begin{cases} \text{match} & \text{if}(x_i = y_j) \\ \text{mismatch} & \text{else} \end{cases}$$

Step4. 最も高いスコアを持つ要素からトレースバックを行う．

文字列 “ACAC” と “AGCA” の類似部分を SW 法で求めたときの文字列テーブルを**図 3** に示す．類似部分を抽出するために，最大のスコアの要素からスコアが 0 の要素までトレースバックを行う．

トレースバックの操作を**図 4** に示す．図 4 において，最大のスコアを持つ要素は $(i,j) = (4,3)$ となり，トレースバックは $(i,j) = (4,3), (i,j) = (3,2), (i,j) = (2,1)$ という経路をたどる．スコアが 0 になるとトレースバックは終了する．トレースバックでは，それぞれの要素がスコアを計算するさいに使用した要素を逆向きに辿り，スコアが 0 のところは含まずに，要素の座標に対応している文字列から類似部分列を抽出する．“ACAC” と “AGCA” からは “CA” が類似度の高い部分文字列として抽出される．

3. 3 次元 Smith-Waterman 法

3.1 概　　要

3 つ以上の文字列の類似部分列を求める手法には様々なものがあるが，そのほとんどが近似法を用いており，複数の類似部分列を抽出することができない．複数の類似部分列を取得するには SW 法が適しているが，2 つの文字列の比較が限度である．本研究報告では，SW 法で 3 つの文字列の比較を行えるように，3 次元 SW 法を作成する．3 次元 SW 法で

Vol.2012-ARC-199 No.8
2012/3/28

X \ Y	-	A	C	A	C
-	0	0	0	0	0
A	0				
G	0				
C	0				
A	0				

図 3 計算後の文字列テーブル

X \ Y	-	A	C	A	C
-	0	0	0	0	0
A	0	1	0	1	0
G	0	0	0	0	0
C	0	0	1	0	1
A	0	1	0	2	0

図 4 トレースバック

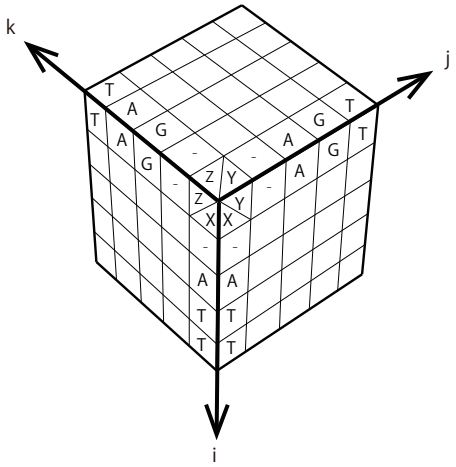

図 5 3 次元 SW 法の文字列テーブル

は，SW 法の文字列テーブルを 3 次元に拡張したものを用いる．**図 5** に 3 次元 SW 法の文字列テーブルを示す．文字列テーブルの各要素の計算を行ったのち，2 次元のときと同様に 3 次元的にトレースバックを行う．文字列テーブルの計算方法およびパラメータの値は 2 章と同じものを用いる．

3.2 計算手順

長さがそれぞれ l，m，n である配列 $X = x_1x_2...x_l$，$Y = y_1y_2...y_m$，$Z = z_1z_2...z_n$ を対象としたときの 3 次元 SW 法のアルゴリズムの流れを以下に示す．

Step1. 配列 X，Y，Z について，文字列テーブルを作成し，それぞれの文字列を i，j，k 軸に割り当てる．

Step2. 位置 i, j, k にある要素が $SW(i, j, k)$ であるとすると，$0 \leq p \leq l$，$0 \leq q \leq m$ および $0 \leq r \leq n$ において，$SW(p, q, r) = SW(p, 0, 0) = SW(0, q, 0) = SW(0, 0, r) = 0$ として初期化を行う．

Step3. それぞれの要素について，式（2）を基にスコアを計算する．

$$SW(i,j,k) = \max \begin{cases} SW(i-1,j-1,k-1) + \mathrm{A} \\ SW(i-1,j-1,k) + \mathrm{gap} \\ SW(i,j-1,k-1) + \mathrm{gap} \\ SW(i-1,j,k-1) + \mathrm{gap} \\ SW(i-1,j,k) + \mathrm{gap} \\ SW(i,j-1,k) + \mathrm{gap} \\ SW(i,j,k-1) + \mathrm{gap} \\ 0 \end{cases} \tag{2}$$

$$A : \begin{cases} \mathrm{A：match} & \mathrm{if}(x_i = y_j = z_k) \\ \mathrm{B：mismatch} & \mathrm{else} \end{cases}$$

Step4. 最も高いスコアを持つ要素からトレースバックを行う．

4. CUDA

4.1 CUDA の概要

本研究報告では，NVIDIA 社が提供している GPU 向けの統合開発環境である CUDA を用いた．CUDA を用いることで GPU を並列計算機として利用することができる．CUDA では，C 言語の構文に加えて，ホストとデバイス（GPU とビデオメモリ）のメモリ確保，解放，データ転送といったデバイスの操作に関する機能の拡張が行われている．

CUDA では，ホストとデバイスの処理を分割して記述する．デバイスの処理は kernel 関数と呼ばれる関数に記述して行い，それ以外をホストで処理する．CUDA によってデバイスで処理を行う際の手順を述べる．まず，(1) ホストでデバイスのメモリ確保を行う．そして，(2) ホストのメモリからデバイスのメモリへ計算に必要なデータを転送する．次に，(3) デバイスの kernel 関数で転送されたデータを処理する．さらに，(4) デバイスのメモリからホストのメモリへ計算結果のデータを転送する．最後に，(5) ホストでデバイスのメモリ解放を行う．このように，ホストで処理を行いつつデバイスに処理を与えている．

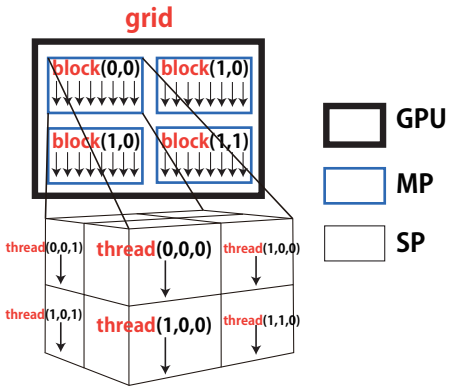

図 6 CUDA における GPU の構成

4.2 デバイスの構成

CUDA を用いたときの GPU の構成を**図 6** に示す．CUDA を利用すると GPU は **grid**，**block**，**thread** の 3 つの単位で管理される．複数の **thread** の集まりを **block** といい，複数の **block** の集まりを **grid** と定義する．1 つの **grid** は 1 つの GPU に対応する．GPU は複数の Streaming Multi Processor(以降 SM) からなり，1 つの SM は 1 つの **block** に対応する．SM は複数の Streaming Processor(以降 SP) からなり，1 つの SP は 1 つの **thread** に対応している．各 **thread** に割り当てられた処理は同時に実行され，並列処理が実現される．

CUDA では実行の際に Warp という単位で同じ命令が行われ同時に処理される．Warp は 32 個の **thread** からなるため，**thread** 数は 32 の倍数の場合が望ましい．Warp 内の各 **thread** が分岐命令でそれぞれが違う方向に分岐を繰り返していくと，並列処理が行われないため GPU の性能低下を招く．このことを Warp divergent といい，GPU で処理を行う際には注意しなければならない．

5. 関 連 研 究

5.1 配列アラインメント

配列アラインメントには，2 つの配列を比較するペアワイズ配列アラインメント (pair-wise sequence alignment) と，3 つ以上の配列を比較する多重配列アラインメント（multiple sequence alignment）がある．また，比較サイズの大きさの面での分類では，配列の全体が含まれるように比較を行う大域的配列アラインメント（global sequence alignment）と配列の一部で比較を行う局所的配列アラインメント（local sequence alignment）がある．配列アラインメントのうち SW 法は精度を重視するが，精度を犠牲にして速さを重視する手法には BLAST(Basic Local Alignment Search Tool)[7] や FASTA(FAST-ALL)[3] がある．

多重配列アラインメントを求める方法には，ペアワイズ配列アライメントで使用している DP 法を 3 つの配列に拡張する手法がある[8]．DP 法の拡張を行う手法では，3 配列以上の比較において，必要な計算ステップ数やメモリ量は解析する配列の数に対して，指数関数的に増加するため，文字列長が短い配列しか扱えないという問題がある．よって，一般的に多重配列アラインメントの計算には近似法が用いられている．

多重配列アラインメントのほとんどは近似法を用いている[9][10][11][12]．近似法の多重配列アラインメントは近似法を用いていない手法に比べて，配列アラインメントの精度が劣る．また，大域的配列アラインメントであるため，類似度が最大のもの以外の類似部分を抽出することができない．それに対して，SW 法は局所的配列アラインメントであるため，n 次元に拡張しても類似度が最大のもの以外の多重配列アラインメントを行うことができる．

5.2 配列アラインメントの GPU 実装

配列アラインメントを GPU で高速化する研究[13][14][15][16] も多く行われている．SW 法の GPU 実装の多くは，SW 法の文字列テーブルを全て保持しておらず，小さなブロックに分割して保持している場合が多い．これは，SW 法の文字列テーブルはデータ量が膨大になるため，デバイス側からホスト側に文字列テーブルを戻すときのデータ転送に時間がかかるためである．また，多くの実装ではデバイス側で文字列テーブルの計算を実行して，ホスト側でトレースバックを行っている．

5.3 並列計算機向けの SW 法

並列計算機に特化した SW 法には Smith-Waterman-Gotoh（SWG）[17] や Striped Smith-Waterman（SSW）[18] などがある．SWG は，FPGA や GPU などの CPU に比べて複雑な命令処理が苦手なアーキテクチャ向けのアルゴリズムである．一方，SSW は，CPU や Cell.B.E. などの複雑な命令処理が比較的得意であるアーキテクチャ向けのアルゴリズムである．

6. 実　　装

6.1 概　　要

fNIRS の脳血流量のデータを 3 次元 SW 法を用いて解析を行う．SW 法は文字列の比較

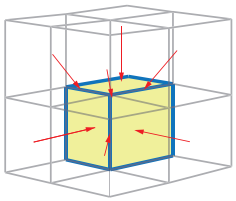

図 7 各要素のデータの依存性

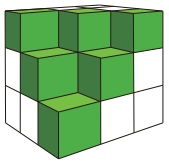

図 8 依存性のない要素

のみを行うので，fNIRS の脳血流量データを文字列に変換し，SW 法を利用できるようにする．SW 法の文字列テーブルをそのまま計算すると時間がかかるので，CUDA を用いて GPU で 3 次元 SW 法を実行する．なお，本研究報告で用いた 3 次元 SW 法はトレースバックの実装を行っていない．

6.2 時系列データから文字列データへの変換

fNIRS の出力データのような時系列データを，文字列データに変換する手法としては，SAX（Symbolic Aggregation approXimation）[19] や等間隔領域分割などがある．時系列データを文字列に変換することで，相同性検索を行うことが可能となる．本研究報告では，SAX を用いて fNIRS の脳血流データを文字列に変換し，時系列データの類似部分抽出が行えるようにする．

6.3 GPU 実装の概要

3 次元 SW 法を下記に示す方法で実装し，それぞれを比較し評価する．ここでは，3 次元 SW 法の文字列テーブルを 4 文字毎に区切ったテーブルを，サブテーブルと定義する．

- サブテーブル使用 GPU 版 3 次元 SW 法（GPUI）
- サブテーブル不使用 GPU 版 3 次元 SW 法（GPUN）
- サブテーブル不使用 CPU 版 3 次元 SW 法（CPU）

図 7 に 3 次元 SW 法の文字列テーブルにおける各要素のデータの依存性を示す．また，**図 8** に 3 次元 SW 法の文字列テーブルの斜めの面において依存性のない要素を示す．3 次元 SW 法の文字列テーブルにおいて，それぞれの要素は図 7 に示すように 7 近傍の要素とデータが依存関係にある．そのため，図 8 に示すように斜めの面において，各要素は依存性を持たず，並列に計算が可能である．また，図 8 に各面ごとに計算可能な要素を示す．このような SW 法の並列性を利用して CUDA で 3 次元 SW 法の GPU 実装を行う．

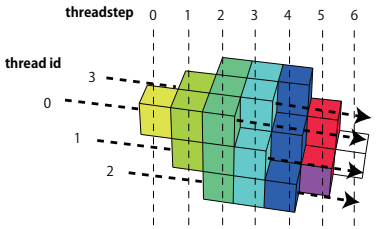

図 9 thread の計算手順

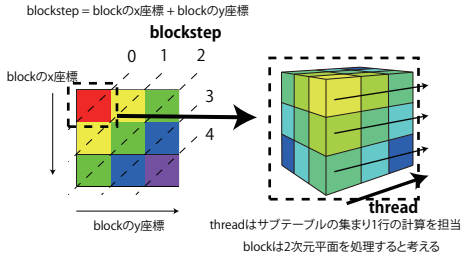

図 10 block の計算手順

6.4 GPU での実装の手順

6.4.1 文字列テーブルの thread, block への割り当て

CUDA における **thread** を**図 9** のように割り当てる．一つの **thread** で文字列テーブルの一辺を全て計算する．図 9 では，step=3 のときに最大 7 つのスレッドが並列に実行されている．また，一つの **thread** で，3 次元 SW 法の文字列テーブルの一辺を全て計算するため，**block** は，文字列テーブルの次元数が減少し，平面上に **block** を割り当てることと同等になる．**図 10** に示すように，**block** の x 座標と y 座標を足したものが現在の step 数と等しいときに，そのブロックが実行される．

6.4.2 サブテーブルの作成

文字列テーブルの計算を高速化するために，**図 11** に示すように，文字列テーブルの要素を $4 \times 4 \times 4$ の 64 要素毎に区切ったサブテーブルを作成し，割り当てた **thread** でそれぞれ処理していく．

通常，デバイス側でメモリを確保すると global memory が確保される，global memory で文字列テーブルをすべて計算すると，メモリへのアクセスに時間がかかるため，アクセス時間が比較的高速である shared memory を用いる．一方，shared memory はメモリ容量が global memory より小さく，格納するデータの大きさが制限される．よって，global memory よりアクセスの高速な shared memory で，64 要素という比較的容量の小さいサブテーブルを用いて部分的に文字列テーブルを計算する．GPU は条件分岐処理において処理が低下するので，64 要素の値の計算をすべて論理演算を使用し条件分岐を削減した．

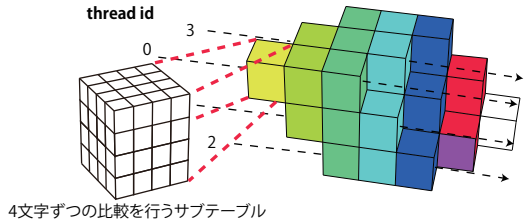

図 11 サブテーブルの作成

7. 評　　価

7.1 概　　要

3 次元 SW 法を fNIRS の脳血流量データに基づいて，GPU と CPU で実行時間について性能評価を行う．fNIRS のデータから生成した 3 つの文字列を，3 次元 SW 法にかけて，文字列テーブルを計算し，CUPS（Cell Updates Per Second：一秒間に文字列テーブルの要素がいくつ計算できるか）について比較検討を行う．SW 法の性能評価の指標として CUPS が一般的に用いられている．比較される文字列の長さを $|A|$，比較する文字列の長さの合計を $|B|$，実行時間を t 秒とすると，GCUPS（Giga CUPS）は，$(|A| \times |B|) \div (t \times 10^9)$ で表される．fNIRS のデータの個数，すなわち文字列長が 8，16，128，256，512 のときについてそれぞれ SW 法で文字列テーブルを計算する．本研究報告では文字列テーブルを計算し，ホスト側に文字列テーブルを戻してくるまでの時間を実行時間とする．

3 次元 SW 法を表 1 の構成のマシンを用いて評価を行う．

7.2 GPU 実装の評価

GPU で文字列テーブルを分割して共有メモリに格納し計算を行った場合（GPUI），GPU でサブテーブルを使用しない場合（GPUN），CPU でサブテーブルを使用しない場合（CPU）の 3 次元 SW 法について評価を行った．結果を**図 12** に示す．

表 1 使用マシン構成

CPU	Intel Core i5-2400 3.10GHz
GPU	GeForce GTX 460
Memory	8GB
OS	Ubuntu 11.04
開発環境	CUDA 4.0
コンパイルオプション	-O3

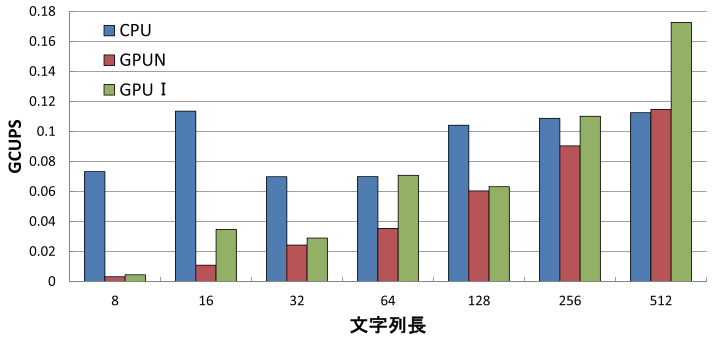

図 12 3 次元 SW 法の評価結果

評価の結果，GPUI では CPU に比べて最大 1.53 倍，GPUN では CPU に比べて最大 1.01 倍，GPUI は GPUN に比べて最大 1.50 倍の性能向上がみられた．また，GPUI は最大で 0.172GCUPS を示した．また最低で 0.004GCUPS を示した．

図 12 より，文字列長が短い程 CPU の方が高速な処理を行っていることがわかる．これは，文字列テーブルの計算時間より，ホスト-デバイス間のデータの転送に時間を要することによる．また，文字列長が長くなるにつれ，CPU に比べて GPU 実装の方が処理性能が高くなることがわかる．

GPUN に比べて GPUI の方が文字列長を長くした場合の，処理性能の向上率が大きいこともわかる．これは，文字列長が増えるほど文字列テーブルへのメモリアクセスが増加するという性質をもつ SW 法を実行ときのアルゴリズムの違いから発生することである．GPUI はサブテーブルに分割して一度に複数のデータを，メモリアクセスの速い shared memory で計算し，値の格納をまとめて行うのに対して，GPUN では文字列テーブルを計算する度，メモリアクセスの遅い global memory からデータを取得しているためである．

次に，GPUN について GPU の枚数と性能の関係を**図 13** に示す．2 枚の GPU についてそれぞれ同じ文字列の比較を行っている．評価の結果 GPU を 2 枚にしたところ，GPU1 枚のときに比べて約 1.8 倍の性能向上がみられた．図 13 より，文字列長によって性能は変わらないことがわかった．また，単純に処理性能が 2 倍とならないのは，転送にかかるオーバーヘッドによるものと考えられる．

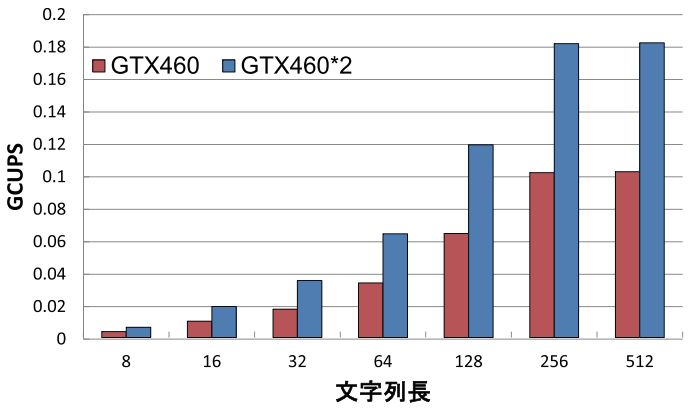

図 13 GPUN における GPU の枚数による 3 次元 SW 法の評価結果

8. まとめと今後の展開

本研究報告では，3 次元 SW 法の提案を行い，fNIRS の脳血流データに基づいて評価を行った．結果として工夫ありの GPU 実装は工夫なしの CPU の実装に比べて最大 1.53 倍の性能向上がみられた．また，2 枚の GPU を用いると文字列長の長さに関係なく約 1.8 倍の性能向上がみられた．

今後の展開としては並列計算機向けの SW 法の導入による処理の高速化，また，さらに GPU の枚数を増やして性能の変化を検討することが考えられる．

参 考 文 献

1) Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Kruger, J., Lefohn, A. and Purcell, T.J.: A Survey of General-Purpose Computation on Graphics Hardware, *Eurographics 2005, State of the Art Reports*, pp.21–51 (2005).
2) Smith, T. and Waterman, M.: Identification of Common Molecular Subsequences, *Journal of Molecular Biology*, Vol.147, No.1, pp.195–197 (1981).
3) Mount, D.W.: バイオインフォマティクス ゲノム配列から機能解析へ第 2 版，メディカル・サイエンス・インターナショナル (2005).
4) 松下晋，中川匡弘：光トポグラフィーによる感性情報解析 (ヒューマンコミュニケーション)，電子情報通信学会論文誌. A, 基礎・境界， Vol.88, No.8, pp.994–1001 (2005-08-01).
5) 廣安知之，西井琢真，吉見真聡，三木光範，横内久猛：相同性検索を用いた 2 つの時系列データからの類似部分抽出手法と DTW による類似部分の評価，情報処理学会研究報告. MPS, 数理モデル化と問題解決研究報告， Vol.2010, No.24, pp.1–6 (2010-09-21).
6) Henikoff, S. and Henikoff, J.: Amino acid substitution matrices from protein blocks., *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, Vol.89, pp.10915–10919 (1992).
7) NCBI: BLAST Basic Local Alignment Search Tool (2009).
8) Liu, Y., Schmidt, B. and Maskell, D.L.: MSA-CUDA: Multiple Sequence Alignment on Graphics Processing Units with CUDA, *Application-Specific Systems, Architectures and Processors, IEEE International Conference on*, Vol.0, pp.121–128 (online), DOI:http://doi.ieeecomputersociety.org/10.1109/ASAP.2009.14 (2009).
9) Thompson, J.D., Higgins, D.G. and Gibson, T.J.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice, *Nucleic Acids Research*, Vol.22, No.22, pp.4673–4680 (1994).
10) Katoh, K., Misawa, K., Kuma, K. and Miyata, T.: MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform, *Nucleic Acids Research*, Vol.30, No.14, pp.3059–3066 (2002).
11) Bray, N. and Pachter, L.: MAVID: constrained ancestral alignment of multiple sequences, *Genome Research*, Vol.14, No.4, pp.693–699 (2004).
12) Notredame, C., Higgins, D.G. and Heringa, J.: T-Coffee: A Novel Method for Fast and Accurate Multiple Sequence Alignment, *Journal of Molecular Biology*, Vol.14, No.4, pp.693–699 (1994).
13) 土肥慶亮， 柿本雄，柴田裕一郎， 濱田剛， 小栗清：GPU クラスタにおける Smith-Waterman アルゴリズムの実装手法の提案，*SACSIS2010*, Vol.2010, No.5, pp.209–216 (2010).
14) 宗川裕馬，伊野文彦，荻原兼一：統合開発環境 CUDA を用いた GPU での配列アライメントの高速化手法，情報処理学会研究報告， Vol.114, No.19, pp.13–18 (2008).
15) de OSandes, E.F. and de Melo, A. C. M.A.: Smith-Waterman Alignment of Huge Sequences with GPU in Linear Space, *2011 IEEE International Parallel & Distributed Processing Symposium*, Vol.25, pp.1199–1211 (2011).
16) Liu, Y., Schmidt, B. and Maskell, D. L.: CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions, *BMC Research Notes*, Vol.3, No.1, pp.1–12 (2010).
17) Gotoh, O.: An Improved Algorithm for Matching Biological Sequences, *Journal of Molecular Biology*, Vol.162, pp.705–708 (1981).

Vol.2012-ARC-199 No.8
2012/3/28

18) Farrar, M.: Striped Smith–Waterman speeds database searches six times over other SIMD implementations, *Bioinformatics*, Vol.23, pp.156–161 (2007).
19) Lin, J., Keogh, E., Lonardi, S. and Chiu, B.: A Symbolic Representation of Time Series, with Implications for Streaming Algorithms, *In Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, ACM Press, pp.2–11 (2003).